Computing the Initial Requirements in Conditioned Behavior Trees

Eleonora Giunchiglia

DIBRIS, Università degli Studi di Genova, Italy

1 Behavior Trees

Following [1], informally a BT is a directed rooted tree in which the nodes are classified as root, control flow nodes or execution nodes; control flow nodes are always internal nodes, and the execution nodes are always leaves. The execution of every BT starts from the root, which sends ticks with a given frequency to its children which can return *Running*, if its execution is not yet completed, *Success*, if it has achieved its goal, and *Failure* otherwise. There are two types of execution nodes (action and condition) and four types of control flow nodes (sequence, fallback, parallel and decorator):

1. Action node: when it receives a tick from its parent it returns *Success* if the action is successfully completed, *Failure* if the action cannot be completed and *Running* otherwise.
2. Condition node: when it receives a tick from its parent, it returns *Success* if the condition is satisfied and *Failure* otherwise. It cannot return *Running*.
3. Sequence node($\rightarrow$): when it receives a tick from its parent it sends it to its children in succession, returning *Failure* (*Running*) as soon as one of them returns *Failure* (*Running*), and *Success* only when all the children have already returned *Success*.
4. Fallback node(?): when it receives a tick from its parent it sends it to its children in succession, returning *Success* (*Running*) as one of them returns *Success* (*Running*), and *Failure* only if all the children return *Failure*.
5. Parallel node($\rightrightarrows$): when it receives a tick from its parent it sends it to its N children in parallel and returns *Success* if a given number M of children returns *Success*, *Failure* if $N - M + 1$ return *Failure* and *Running* otherwise. In this paper we suppose that a parallel returns *Success* if all N children return *Success*, and that all its children are action nodes.

If we define a plan as a sequence of set of actions, then we understand that the policy defined by the BT is the set of plans that are compliant with the BT itself, and which can be derived from the rules we have given above.

2 Conditioned Behavior Trees

Intuitively, a CBT extends classical Behavior Trees because for each action it specifies both the action performed, and the set of preconditions which must be satisfied in order to perform the action and the set of postconditions that are necessarily satisfied if the action is successful. More formally, we define a CBT over two sets:

1. the set of boolean variables $\mathcal{A}$ representing the actions that the agent may perform on the environment,
2. the set of literals $\mathcal{C}$ representing the state of the agent and the state of the environment in which the agent operates.

Hence, given the sets $\mathcal{A}$, $\mathcal{C}$, and the designer can define a function $cond : \mathcal{A} \to \mathcal{P}(\mathcal{C}) \times \mathcal{P}(\mathcal{C})$ such that it associates to each action $a \in \mathcal{A}$ a couple $(Pre,\ Post)$ in which:

1. $Pre \subseteq \mathcal{C}$ represents the set of preconditions, and
2. $Post \subseteq \mathcal{C}$ represents the set of postconditions.

When modeling the CBT we make the following additional assumptions:

1. each action is instantaneous,
2. each pre- and post- condition $c \in \mathcal{C}$ is modeled as either a propositional variable or its negation,
3. given an action a it cannot be the case that $cond(a) = (Pre, Post)$ such that there exist $c \in Pre$ and $\neg c \in Pre$,
4. given an action a it cannot be the case that $cond(a) = (Pre, Post)$ such that there exist $c \in Post$ and $\neg c \in Post$,
5. if two actions a and a' are children of the same parallel node, then it cannot be the case that $cond(a) = (Pre_a, Post_a)$ and $cond(b) = (Pre_b, Post_b)$ such that there exist $c \in Post_a$ and $\neg c \in Post_b$.

As it can be seen in Figure 1, given an action a, we can represent it in our CBT as an action node with on top the label "a" and below the sets Pre and $Post$ such that $cond(a) = (Pre,\ Post)$. Further, in CBTs, we model any condition node having condition $c \in \mathcal{C}$ by associating to it a dummy action a, $Pre = \{c\}$ and $Post = \emptyset$.

3 Computing the initial requirements

3.1 Building the state graph

Given the sets $\mathcal{A}$ and $\mathcal{C}$, we can build a state graph $\langle \mathcal{S}, \mathcal{T} \rangle$ which represents all the possible plans carried out by the agent and in which:

- $\mathcal{S}$ represents the set of possible states given $\mathcal{C}$ and it is computed as $\mathcal{S} = \mathcal{P}(\mathcal{C})$.
- $\mathcal{T}$ represents the transition function $\mathcal{T} : \mathcal{S} \times \mathcal{P}(\mathcal{A}) \to \mathcal{S}$ such that for each state $s \in \mathcal{S}$ and set of actions $A \in \mathcal{P}(\mathcal{A})$ it returns a new state s'. Given s and A we will be able to perform the set of actions A if for every action $a \in A$ we have that $cond(a) = (Pre_a, Post_a)$ such that:
  - if $c \in Pre_a$ then $c \in s$, and

- if $\neg c \in Pre_a$ then $c \notin s$.

Given s and A the new state s' will be equal to:

$$s' = s \cup \{c \mid c \in Post_a, a \in A\} \setminus \{c \mid \neg c \in Post_a, a \in A\}.$$

Given the above definitions, we can notice that a CBT always defines finite plans, and, hence, it is always possible to build a representation of all these plans as a propositional logic formula. Further, it is also possible to get such representation of the above state graph. As we will show below, those representations can be obtained in polynomial time with respect to the number of nodes in the CBT, and, if put in conjunction, give us the ability of computing the initial requirements necessary to successfully complete at least one of the plans associated to the CBT.

3.2 Transition System Representation

Suppose that the longest sequence of set of actions has length equal to N. Given the above, and given the sets $\mathcal{A}$ and $\mathcal{C}$, we can represent the transition system as a conjunction of the below formulas:

- for every $a \in \mathcal{A}$ such that $cond(a) = (Pre_a, Post_a)$ then:

$$a_i \rightarrow \bigwedge\{c_i \mid c \in Pre_a\}, \tag{1}$$

$$a_i \rightarrow \bigwedge\{c_{i+1} \mid c \in Post_a\} \tag{2}$$

- for every condition $c \in \mathcal{C}$ then:

$$(c_i \wedge \neg c_{i+1}) \rightarrow \bigvee\{a_i \mid \neg c \in Post_a\}, \tag{3}$$

$$(\neg c_i \wedge c_{i+1}) \rightarrow \bigvee\{a_i \mid c \in Post_a\} \tag{4}$$

for $i = 0, \ldots, N-1$.

3.3 Behavior Tree Plans Representation

Once we have represented the transition system, we have to define which are the plans described by our CBT. In order to build the propositional formula we can follow the pseudocode in Algorithm 1. In the algorithm we assume that given any node X then its children x_i for $i, \ldots, M$, M being the total number of children of X, always define sequences of actions of the same length. If this assumption is violated we can make it hold through the substitution from the leaves to the root of each node x_i with a sequence node having as first $L_{max} - L_i$ children dummy action nodes $NoOps$ and as L_{max} node the node x_i itself. In the above we have called L_{max} the length of the longest sequence of actions defined by one of the children of X and L_i the length of the sequence of each child x_i while with $NoOps$ we indicate an action node such that $Pre = \emptyset$ and $Post = \emptyset$. This is a sensible operation to perform because we are not actually interested in "when" the action is performed, but we are only interested in giving a total order among the actions in the CBT.

Algorithm 1 Get Propositional Formula from CBT

```
function getPropositionalFormula( )
    return visit(rootBT, 0, root)
function visitBT(node, t)
    propFormula = []
    if node is execution_node : then
        return a_t ⋀{¬b_t | b ∈ A \ a}
    else
        propFormula.append( '(' )
        if node is fallback_node : then
            for each child in children do
                propFormula.append(visit(child, t))
                if node.hasOtherChildren then
                    propFormula.append( ∨ )
        if node is sequence_node : then
            for each child in children do
                propFormula.append(visit(child, t))
                t = t +1
                if node.hasOtherChildren then
                    propFormula.append( ∧ )
        if node is parallel_node : then
            for each a in children.actions do
                propFormula.append(a_t)
                if node.hasOtherChildren then
                    propFormula.append( ∧ )
            for each a in (A \ children.actions) do
                propFormula.append( ∧ )
                propFormula.append(¬a_t)
        propFormula.append( ')' )
    return propFormula
```

4 Example

To make more intuitive what we have explained in the above we build the representations associated to the CBT in Figure 1. Since the longest sequence has length $N = 2$, we have that for $i = 0, 1$ the below formulas must hold:

– State graph representation:

$$PassObject_i \rightarrow \neg ObjectNear_i \tag{5}$$

$$PassObject_i \rightarrow ObjectNear_{i+1} \tag{6}$$

$$UseObject_i \rightarrow ObjectNear_i \wedge ObjectFunctioning_i \tag{7}$$

$$UseObject_i \rightarrow \top_{i+1} \tag{8}$$

$$(ObjectNear_i \wedge \neg ObjectNear_{i+1}) \rightarrow \bot_i \tag{9}$$

$$(\neg ObjectNear_i \wedge ObjectNear_{i+1}) \rightarrow PassObject_i \tag{10}$$

$$(ObjectFunctioning_i \wedge \neg ObjectFunctioning_{i+1}) \rightarrow \bot_i \tag{11}$$

$$(\neg ObjectFunctioning_i \wedge ObjectFunctioning_{i+1}) \rightarrow \bot_i \tag{12}$$

– CBT plans representation:

$$\begin{aligned}&(PassObject_0 \wedge \neg UseObject_0 \wedge UseObject_1 \wedge \neg PassObject_1)\\ &\vee (\neg PassObject_0 \wedge \neg UseObject_0 \wedge UseObject_1 \wedge \neg PassObject_1)\end{aligned} \tag{13}$$

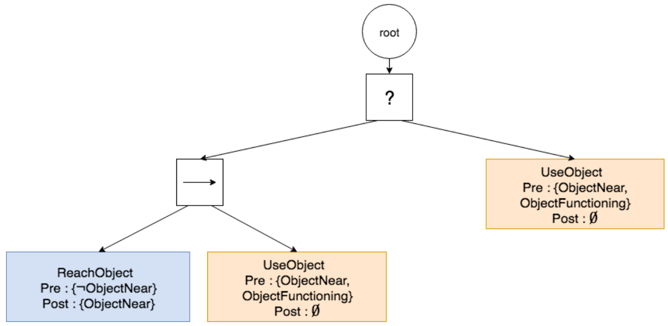

Fig. 1. Example of CBT.

If we check the satisfiability of the conjunction of the above formulas we get that the initial requirements are:

$$(\neg ObjectNear_0 \wedge c_0) \vee (ObjectNear_0 \wedge c_0) \equiv c_0$$

References

1. Colledanchise, M., Ögren, P.: Behavior trees in robotics and AI: an introduction. CoRR abs/1709.00084 (2017), http://arxiv.org/abs/1709.00084